ISSN No. 0976-5697

Volume 3, No. 6, Nov. 2012 (Special Issue)

International Journal of Advanced Research in Computer Science

CASE STUDY AND REPORT

Available Online at www.ijarcs.info

A Survey on Process Migration

Vinita Sharma*
M.Tech Scholar, Dept of CSE,
KIET, Ghaziabad

Dr. Vineet Sharma
Associate Professor, Dept of CSE,
KIET, Ghaziabad

Abstract: The days of supercomputers and mainframes dominating computing are over. With the cost benefits of the mass production of smaller machines, the majority of today's computing power exists in the form of PCs or workstations, with more powerful machines performing more specialized tasks. Even with increased computer power and availability, some tasks require more resources. Load balancing and process migration allocates processes to interconnected workstations on a network to better take advantage of available resources.
Process migration is the act of transferring a process between two machines during its execution. It enables dynamic load distribution, fault resilience, eased system administration, and data access locality. With the increasing deployment of distributed systems in general, and distributed operating systems in particular, process migration is receiving more attention in both research and product development. As high-performance facilities shift from supercomputers to networks of workstations, and with the ever-increasing role of the World Wide Web, we expect migration to play a more important role and eventually to be widely adopted.
This paper reviews the field of process migration by summarizing the key concepts involved in it and by highlighting the benefits and challenges faced by the process migration. It also includes the experience of process migration in distributed operating system.

Keywords: process migration, fault resilience, load distribution, distributed systems.

I. INTRODUCTION

In a network of personal workstations, many machines are typically idle at any given time. These idle hosts represent a large deposited of processing power, many times it is greater than the available on any user's personal machine in isolation. In recent years a number of mechanisms have been proposed or implemented to bring under conditions for effective use of idle processors .Here we are considering process migration mechanism for this purpose.

A distributed operating system is the logical aggregation of operating system software over a collection of independent, networked, communicating, and physically separate computational nodes. [22] Individual nodes each hold a specific software subset of the global aggregate operating system. Each subset is a composite of two distinct service provisions. [18] The first is a ubiquitous minimal kernel, or microkernel, that directly controls that node's hardware. Second is a higher-level collection of system management components that coordinate the node's individual and collaborative activities. These components abstract microkernel functions and support user applications.[6]An example of an distributed operating system in which process migration mechanism is implemented is "Amoeba".Amoeba is a distributed operating system. It collects a huge Varity of single machines connected over a (fast) network to one, huge computer. It was originally developed at the Vrije Universiteit in Amsterdam by Andrew Tanenbaum and many more. Amoeba was always designed to be used, so it was deemed essential to achieve extremely high performance. Currently, it's the fastest distributed operating system. [13]

This paper is organized in 6 sections. Section 1 provides the introduction about the process migration and distributed operating system. Section 2 describes the process migration in detail including homogeneous and heterogeneous types of process migration and steps involved in process migration algorithm. Section 3 explains the benefits of process migration. Section 4 highlights the challenges faced by process migration. Section 5 presents future research. Finally conclusion is given in section 6.

II. OVERVIEW OF PROCESS MIGRATION

A process may be considered as a program in execution. And "Process migration is the act of transferring an active process between two machines and restoring the process from the point it left off on the selected destination node."[1] i.e. transferring an executing process from source machine to destination machine during its execution is called process migration.

If we transfer the state of a process from one machine to another, we have migrated the process. The term state refers to all the information which is required to resume a process in a proper way and executing it correctly.

A. Goals:

The goals of process migration are very closely tied to the applications that use migration, as described next. The primary goals include:

CONFERENCE PAPER
II International Conference on Issues & Challenges in Networking, Intelligence & Computing Technologies
Organized by Krishna Institute of Engineering and Technology (KIET) Ghaziabad, India

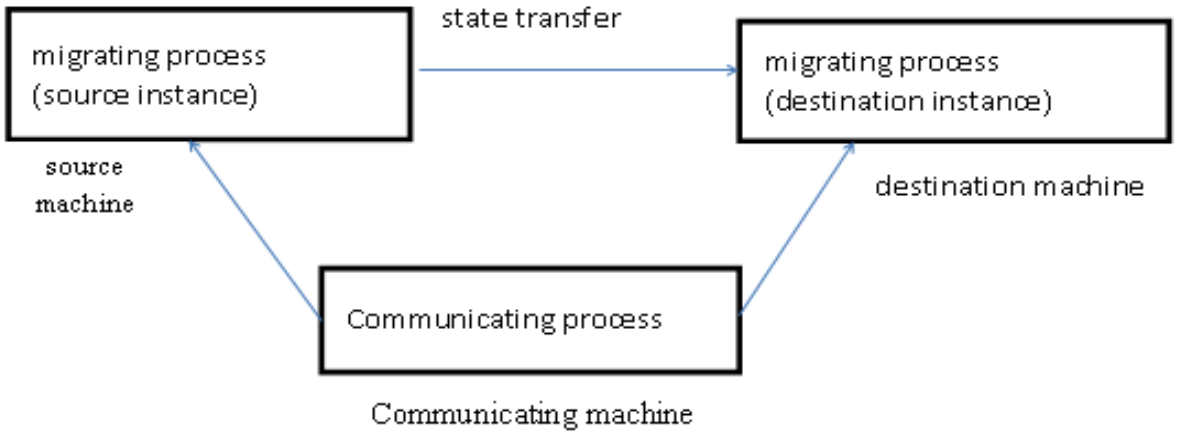

Figure 1: A view of Process Migration

i. Resource locality: Processes which are running on a node which is distant from the node which houses the data that the processes are using tend to spend most of their time in performing communication between the nodes for the sake of accessing the data. Process migration can be used to migrate a distant process closer to the data that it is processing, thereby ensuring it spends most of its time doing useful work.
ii. Resource sharing: Nodes which have large amount of resources can act as receiver nodes in a process migration environment.
iii. Effective Load Balancing: Migration is particularly important in receiver initiated distributed systems, where a lightly loaded node announces its availability thereby enabling the arbitrator to provide its processing power to another node which is relatively heavily loaded.
iv. Fault tolerance: This aspect of a system is improved by migration of a process from a partially failed node, or in the case of long running processes when different kinds of failures are probable. In conjunction with check pointing, this goal can be achieved.
v. Eased system administration: When a node is about to be shutdown, the system can migrate processes which are running on it to another node, thereby enabling the process to go to completion, either on the destination node or on the source node by migrating it back.
vi. Mobile computing: Users may decide to migrate a running process from their workstations to their mobile computers or vice versa to exploit the large amount of resources that a work station can provide.

B. Applications:

The following applications can be benefited from process migration:

i. Distributed applications can be started on certain nodes and can be migrated at the application level or by using a system wide migration facility in response to things like load balancing considerations.
ii. Multiuser Applications can benefit greatly from process migration. As users come and go, the load on individual nodes varies widely. Dynamic process migration can automatically spread processes across all nodes, including those applications that are not enhanced to exploit the migration mechanism.
iii. Standalone Applications, which is pre emptable, can be used with various goals in mind. Such an application can either migrate itself, or it can be migrated by another authority. It is difficult to select such applications without detailed knowledge of past behavior, since many applications are short-lived and do not execute long enough to justify the overhead of migration
iv. Long running applications, which can run for days or weeks on end, can suffer various interruptions, for example partial node failures or administrative shutdowns. Process migration can relocate these processes in the event of the occurrences of any of the events mentioned above.
v. Migration-oriented Applications are applications that have been coded to explicitly take advantage of process migration. Dynamic process migration can automatically redistribute these related processes if the load becomes uneven on different nodes, e.g. if processes are dynamically created, or there are many more processes than nodes.
vi. Mobile applications are the most recent example of the potential use of migration; for instance, mobile agents and mobile objects. These applications are designed with mobility in mind. Although this mobility differs significantly from the kinds of process migration considered elsewhere in this paper, it uses some of the same techniques: location policies, checkpointing, transparency, and locating and communicating with a mobile entity.
vii. Migration-aware applications are applications that have been coded to explicitly take advantage of process migration. Dynamic process migration can automatically redistribute these related processes if the load becomes uneven on different nodes, e.g. if processes are dynamically created, or there are many more processes than nodes.

C. Migration Algorithm [14]:

Although there are many different migration implementations and designs, most of them can be summarized in the following steps:

i. A migration request is issued to a remote node. After negotiation, migration has been accepted.
ii. A process is detached from its source node by suspending its execution, declaring it to be in a migrating state, and temporarily redirecting communication as described in the following step.
iii. Communication is temporarily redirected by queuing up arriving messages directed to the migrated process, and by delivering them to the process after migration. This

step continues in parallel with steps iv, v, and vi, as long as there are additional incoming messages. Once the communication channels are enabled after migration (as a result of step vii), the migrated process is known to the external world.

iv. The process state is extracted, including memory contents; processor state (register contents); communication state (e.g., opened files and message channels); and relevant kernel context. The communication state and kernel context are OS dependent. Some of the local OS internal state is not transferable. The process state is typically retained on the source node until the end of migration, and in some systems it remains there even after migration completes. Processor dependencies, such as register and stack contents, have to be eliminated in the case of heterogeneous migration.

v. A destination process instance is created into which the transferred state will be imported. A destination instance is not activated until a sufficient amount of state has been transferred from the source process instance. After that, the destination instance will be promoted into a regular process.

vi. State is transferred and imported into a new instance on the remote node. Not all of the state needs to be transferred; some of the state could be lazily brought over after migration is completed.

vii. Some means of forwarding references to the migrated process must be maintained. This is required in order to communicate with the process or to control it. It can be achieved by registering the current location at the home node (e.g. in Sprite), by searching for the migrated process (e.g. in the V Kernel, at the communication protocol level), or by forwarding messages across all visited nodes (e.g. in Charlotte). This step also enables migrated communication channels at the destination and it ends step iii as communication is permanently redirected.

viii. The new instance is resumed when sufficient state has been transferred and imported. With this step, process migration completes. Once all ofthe state has been transferred from the original instance, it may be deleted on the source node.

D. Types of Process Migration:

There are two types of process migration:

i. Homogeneous process migration
ii. Homogeneous process migration

a. Homogeneous process migration:

Homogeneous process migration involves migrating processes in a homogeneous environment where all systems have the same architecture and operating system but not necessarily the same resources or capabilities. Process migration can be performed either at the user-level or the kernel level. A brief overview of the two techniques and a few systems that implement them are given in the next sections.

(a) User-level Process Migration

User-level process migration techniques support process migration without changing the operating system kernel. User-level migration implementations are easier to develop and maintain but have two common problems:

i. They cannot access kernel state which means that they cannot migrate all processes.
ii. They must cross the kernel/application boundary using kernel requests which are slow and costly.

User-level process migration facilities vary in complexity from the UNIX rsh command which allows static, remote execution of UNIX commands to Condor which allows dynamic migration of processes using check pointing. Another implementation relies on the cooperation between the process and the migration subsystem to achieve migration. The problem with these implementations is that without kernel access, they are unable to migrate processes with location dependent information and inter process communication.

(b) Kernel Level Process Migration:

Kernel level process migration techniques modify the operating system kernel to make process migration easier and more efficient. Kernel modifications allow the migration process to be done quicker and migrate more types of processes. Unfortunately, many older implementations have high overhead, long freeze times, and still cannot migrate all processes.

b. Heterogeneous process migration:

Homogeneous process migration allows good use of available network resources but is only applicable between machines with the same architecture and operating system. Many networks contain a variety of machines running different operating systems and provide other resources available on machines which have different architectures than the current machine where the process is executing. Using this computational power requires heterogeneous process migration.

Heterogeneous process migration is process migration across machine architectures and operating systems. Obviously, it is more complicated than the homogeneous case because it must consider machine and operating specific structures and features, as well as transmitting the same information as homogeneous process migration including process state, address space, and file and communication information. Heterogeneous process migration is especially applicable in the mobile environment where is highly likely that the mobile unit and the base support station will be different machine types. It would be desirable to migrate a process from the mobile unit to the base station and vice versa during computation. This could not be achieved by homogeneous migration in most cases.

There are 4 basic types of heterogeneous migration [6]:

(a) Passive object- only data is transferred and must be translated.

(b) Active object, migrate when inactive- The process is migrated when it is not executing. The code exists at both sites, and only the data need be transferred and translated.

(c) Active object, interpreted code- The process is executing through an interpreter so only data and interpreter state need be transferred.

(d) Active object, native code- Both code and data need to be translated as they are compiled for a specific architecture.

CONFERENCE PAPER
II International Conference on Issues & Challenges in Networking, Intelligence & Computing Technologies
Organized by Krishna Institute of Engineering and Technology (KIET) Ghaziabad, India

III. CHALLENGES IN PROCESS MIGRATION

The process migration mechanism faces a variety of challenges. Because designing a process migration facility enables the movement of an executing process from one host to another. And hence it includes issues like when and where to migrate which process and achieving the goals of load balancing and transparency with as low overhead as possible presents a big challenge. The main issues related to migration are as follows:

i. The one of the main issue is **Allocation and scheduling** i.e**.** how is a target node chosen? What are the factors taken into consideration while choosing a destination host? Is load balanced dynamically, or only reallocated during special circumstances like eviction or imminent host failure? Considering that all of the above systems represent loosely coupled environments, how much of a difference can such a consideration make? Similarly, what is the best allocation policy for an I/O intensive process?

ii. Once a target has been chosen, **how is the process state saved and transferred?** For e.g., would virtual memory pages be transferred all at once, increasing the latency between process suspension and resumption, or transferred on a demand-paged basis thus speeding up migration? An important consideration over here is how much of "residual dependency" [8] do we allow on the ex-host?

iii. How is migration supported by the underlying **file system** for kernel level schemes? Are files assumed to be accessible from any point? For transparency, a transparent file system would itself seem to be a prerequisite.

iv. How are **name spaces** dealt with? Do process Ids, file descriptors etc. change with migration? How does global naming help? How are sockets and signals managed?

v. What are the **scaling** considerations that have been incorporated into the design?

vi. ***Transparency:*** What is the level of **transparency**? An important goal in process migration is transparency. This means that neither the process being migrated, nor user processes with which it is communicating, should be aware of the migration.

vii. ***Memory Transfer:*** Moving the content of a large virtual address space stands out as the bottleneck in process migration.

viii. ***Residual Dependencies:*** A particular problem in migrating a process is the routing of messages addressed to the migrated process, since the sender of the message need not know about the migration. One way of handling this is for the source machine to redirect messages to the destination machine. This is an example of a residual dependency. In general residual dependencies are undesirable because of the chain of dependencies when a process is migrated several times and the continuing use of resources on the source machine. This has detrimental effects on both performance and reliability.

IV. BENEFITS OF PROCESS MIGRATION

The benefits of process migration are many and varied, especially so nowadays, with the rapid increase in distributed and networking systems. These are discussed below.

i. **Dynamic load distribution** is possible in multiprocessing systems to balance the load on the different processors/nodes, by migrating processes from overloaded nodes to less loaded ones.

ii. **Fault resilience** can be achieved in such systems, by migrating processes from nodes that may have experienced a partial failure or are likely to fail completely in the immediate future.

iii. **Improved system administration** can be achieved by migrating processes from the nodes that are about to be shut down or otherwise made unavailable.

iv. **Data access locality** is possible to provide in wireless or fixed mobile systems, by migrating processes closer to the source of some data as the user moves from one network or cell to another.

v. **Resource sharing** is possible on a grid, by migration of a process to a specific node that is equipped with a special hardware device, large amount of free memory or some other unique resource.

vi. **Mobile computing** also increases the demand for migration. Users may want to migrate running applications from a host to their mobile computer as they connect to a network at their current location or back again when they disconnect.

vii. **Reliability** is achieved through Process migration because it is able to move a copy of a process (replicate) on another node hence improves system reliability.

viii. **Security:** A process dealing with sensitive data may be moved to a secure machine (or just to a machine holding the data) to improve security.

ix. **High performance cluster computing systems** have used process migration to balance the workload on their constituent computers and thus improve their overall throughput and performance

x. **Improved average turnaround time**: turnaround time of a process is total time between submission of a process and its completion. The turnaround time of a job running on a cluster is very important for both the users as well as the system administrators. But any event which causes the job to fail, results in the wastage of all the computations done till that point. The job has to be restarted all over again. This results in increase in the average turnaround time of a job, which is not desirable in a production environment. An approach to address this issue is to transfer the execution context of a running process from a failing to a healthy machine. This is achieved by saving the execution context of a running process at regular intervals of time. The saved execution context is called checkpoint.

V. FUTURE RESEARCH

After around 10 years, process migration is still a field of active research and many of its benefits are yet to be realized. Process migration and load balancing algorithms have been around for quite a while. It is well-known what has to be transferred during migration, so process migration algorithms can only be improved by transmitting this data in the most efficient way.

However, determining when to migrate and measuring processor load still has important research applications. Deciding when to migrate represents the common problem in distributed computing of making decisions with incomplete information. Research into predictive measures

CONFERENCE PAPER
II International Conference on Issues & Challenges in Networking, Intelligence & Computing Technologies
Organized by Krishna Institute of Engineering and Technology (KIET) Ghaziabad, India

of load would be useful. It may be possible to determine future load by examining past requirements and current process properties. Heterogeneous process migration is a lot more interesting research area, but it remains to be proven that it is useful and worth the cost. An interesting benefit of research in this area is improving languages and compilers to provide better migration and heterogeneity support.

An important advanced application of process migration is in the utilization of networks of workstations (NOWs). Many organizations have massive computing power available when workstations are combined over a network. Unfortunately, current operating systems and programs do not take advantage of most of these resources. As an extension to process migration, process division would be useful. Process division would be similar to parallel processing except it is more transparent to the programmer. The goal would be the integration of a programming language/compiler, which allows the programmer to specify what parts of the process can be done in parallel or remotely, and a distributed operating system, which uses this specification and current processor loads to divide the process among various processors (maybe in a heterogeneous fashion) transparently at run-time for the user. This would result in increased machine utilization and some increased parallelism and performance for individual applications.

VI. CONCLUSION

This paper is a survey of process migration. Process migration involves transferring a running process between machines. In homogeneous process migration, this transfer is between machines of the same type, while heterogeneous process migration transfers processes between machines of different architectures and operating systems.

Process migrations are efficient mechanisms to be used in the improvement or development of high performance computer systems. It allow for better utilization of networks of workstations In particular, we demonstrate that the process migration is very crucial to be used to enable dynamic load balancing, excellent system administration, efficient resource utilization, fault resilience, and data access locality. Despite of these primary benefits here we also explained other benefits provided by process migration, and challenges or issues process migration mechanism have to deal with.

VII. REFERENCES

[1] Barak, A. and Litman, A. (August 1985). MOS: a Multicomputer Distributed Operating System. Software – Practiceand Experience, 15(8):725–737.

[2] Chris Steketee, Wei Ping Zhu, and Philip Moseley School of Computer and Information Science, University of South Australia, TheLevelsSA5095, Australia. Chris. Steketee@Unisa.edu.au

[3] Douglis, F. and Ousterhout, J. (September 1987). Process Migration in the Sprite Operating System. Proceedings of the Seventh International Conference on Distributed Computing Systems, pages 18–25.

[4] Douglis, F. and Ousterhout, J. (August 1991). Transparent Process Migration: Design Alternatives and the Sprite Implementation.

[5] D. Eager and E. Lazowska and J. Zahorjan: The Limited Performance Benefits of Migrating Active Processes for Load Sharing. In Conf. on Measurement & Modeling of Comp. Syst., (ACM SIGMETRICS), May 1988, pages 63—72

[6] Distributed Operating Systems: The Logical Design, 1st edition Goscinski, A. 1991 Distributed Operating Systems: the Logical Design. 1st. Addison-Wesley Publishing Co., Inc.

[7] E. H. Baalbergen, Parallel and distributed compilations in loosely-coupled systems: a case study,Proceedings of Workshop on Large Grain Parallelism, Providence, RI, October 1986

[8] Frederick Douglas: Transparent Process Migration in the Sprite Operating System (PhD Thesis,University of California, Berkeley), September 1990.

[9] Fred Douglis and John K. Ousterhout. Transparentprocess migration: Design alternatives andthe Sprite implementation. Software – Practiceand Experience, 21(8):757-785, 1991

[10] F. Douglis and J. Ousterhout, Process migrationin the Sprite operating system, Proceedings ofthe 7th International Conference on DistributedComputing Systems, IEEE, Berlin, West Germany,September 1987, pp. 1825.

[11] F. Douglis and J. Ousterhout: Transparent Process Migration: Design Alternatives and the Sprite Implementation. In Software -- Practice and Experience, volume 21, number 8, pages 757--785, August 1991.

[12] http://en.wikipedia.org/wiki/Distributed_operating_system

[13] http://fsd-amoeba.sourceforge.net/

[14] Dejan S. Milojicic, Fred Douglis, Yves Paindaveine, Richard Wheeler And Songnian Zhou, Process Migration , Hp Labs, At&T Labs–Research, Tog Research Institute, Emc, And University Of Toronto And Platform Computing.

[15] M.Rasit Eskicioglu. design issues in process migration facilities in distributed system, 199, pages 414-424

[16] J.M. Smith. A survey of process migration mechanisms. Technical report, Columbia University, 1995

[17] M. Litzkow, M. Livny, and M. Mutka. Condor – a hunter of idle workstations. In Proceedings of the 8th International Conference on Distributed Computing, 1988.

[18] Nutt, G. J. 1992 Centralized and Distributed Operating Systems. Prentice Hall Press.

[19] Peter Smith and Norman C. Hutchinson. Heterogeneous process migration: The Tui system. Technical Report TR-96-04, UBC Computer Science Department, Vancouver, B.C., February 1996.

[20] Ramon LawrenceDepartment of Computer Science University of Manitobaumlawren@cs.umanitoba.ca, a Survey of Process Migration Mechanisms, May 29, 1998

CONFERENCE PAPER
II International Conference on Issues & Challenges in Networking, Intelligence & Computing Technologies
Organized by Krishna Institute of Engineering and Technology (KIET) Ghaziabad, India

[21] Sunil Thulasidasan University of Southern California, thulasid@usc.edu, Issues in Process Migration December 15, 2000

[22] Tanenbaum, Andrew S. 1993 Distributed operating systems anno 1992. What have we learned so far? Distributed Systems Engineering, 1, 1 (1993), 3-10

[23] Y. Artsy and R. Finkel, Designing a process migration facility: the Charlotte experience, IEEE Computer, 22, (9), 4756 (1989)